Graph representation and elementary algorithms

Data Structures and Algorithms

Tran Ngoc Bao Duy
Faculty of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM

Overview

1. **Graph representation**
   - Adjacency-list representation
   - Adjacency-matrix representation
   - Weighted graphs
2. **Searching a graph**
   - Breadth-first search
   - Depth-first search
3. **Topological sorting**

GRAPH REPRESENTATION

Graph representation

Two standard ways to represent a graph $G = (V, E)$ apply to both directed and undirected graphs:

1. **Adjacency-list:** provides a compact way to represent **sparse** graph $(|E| \ll |V|^2)$.

Graph representation

Two standard ways to represent a graph $G = (V, E)$ apply to both directed and undirected graphs:

1. **Adjacency-list:** provides a compact way to represent **sparse** graph ($|E| \ll |V|^2$).

2. **Adjacency-matrix:** preferred when the graph is **dense** ($|E| \approx |V|^2$) or when need to tell quickly if there is an edge connecting two given vertices.

Adjacency-list representation

Definition

The **adjacency-list representation** of a graph $G = (V, E)$:

- Consists of an array Adj of $|V|$ lists, one for each vertex in V.

Adjacency-list representation

Definition

The **adjacency-list representation** of a graph $G = (V, E)$:

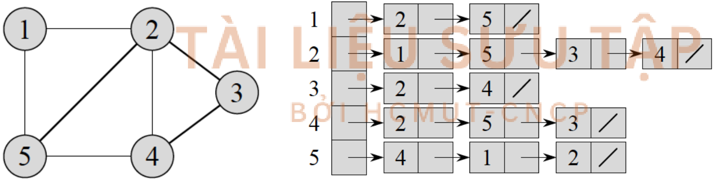
- Consists of an array Adj of $|V|$ lists, one for each vertex in V.
- For each $u \in V$, the adjacency list $Adj[u]$ contains all the vertices such that there is an edge $(u, v) \in E$ or all the vertices adjacent to u in G.

Adjacency-list representation

Definition

The **adjacency-list representation** of a graph $G = (V, E)$:

- Consists of an array Adj of $|V|$ lists, one for each vertex in V.
- For each $u \in V$, the adjacency list $Adj[u]$ contains all the vertices such that there is an edge $(u, v) \in E$ or all the vertices adjacent to u in G.



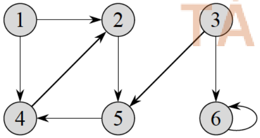
(Source: Introduction to algorithms - 3rd edition)

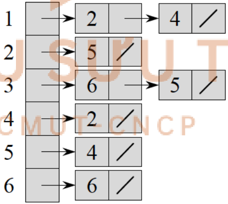
Adjacency-list representation

Definition

The **adjacency-list representation** of a graph $G = (V, E)$:

- Consists of an array Adj of $|V|$ lists, one for each vertex in V.
- For each $u \in V$, the adjacency list $Adj[u]$ contains all the vertices such that there is an edge $(u, v) \in E$ or all the vertices adjacent to u in G.





(Source: Introduction to algorithms - 3rd edition)

Adjacency-list: Properties

1. If G is a **directed** graph, the sum of the lengths of all the adjacency lists is $|E|$.
2. If G is a **undirected** graph, the sum of the lengths of all the adjacency lists is $2|E|$.
3. The amount of memory required for both cases of graph is $\Theta(V + E)$.

Adjacency-list: Properties

1. If G is a **directed** graph, the sum of the lengths of all the adjacency lists is $|E|$.
2. If G is a **undirected** graph, the sum of the lengths of all the adjacency lists is $2\,|E|$.
3. The amount of memory required for both cases of graph is $\Theta(V + E)$.

Disadvantages: provides no quicker way to determine whether a given edge (u, v) is present in the graph than to search for in the adjacency list $Adj[u]$.

Adjacency-matrix representation

Definition

Assume that the vertices are numbered $1, 2, \ldots, |V|$ in some arbitrary manner, the **adjacency-matrix representation** of a graph $G = (V, E)$ consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Adjacency-matrix representation

Definition

Assume that the vertices are numbered $1, 2, \ldots, |V|$ in some arbitrary manner, the **adjacency-matrix representation** of a graph $G = (V, E)$ consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

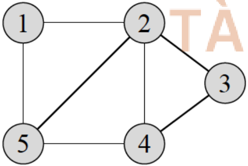
$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Adjacency-matrix representation

Definition

Assume that the vertices are numbered $1, 2, \ldots, |V|$ in some arbitrary manner, the **adjacency-matrix representation** of a graph $G = (V, E)$ consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$



$$\begin{array}{c|ccccc|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array}$$

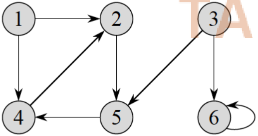
(Source: Introduction to algorithms - 3rd edition)

Adjacency-matrix representation

Definition

Assume that the vertices are numbered $1, 2, \ldots, |V|$ in some arbitrary manner, the **adjacency-matrix representation** of a graph $G = (V, E)$ consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(Source: Introduction to algorithms - 3rd edition)

Adjacency-matrix: Properties

1. Requires $\Theta(V^2)$ memory, independent of the number of edges in the graph.

2. Along the main diagonal of the adjacency matrix, there is a symmetry such that the adjacency matrix A of an undirected graph is its own transpose: $A = A^T$.

Weighted graphs

To represent **weighted graphs**, that is, graphs for which each edge has an associated **weight**, typically given by a weight function $w : E \to \mathbb{R}$.

- Adjacency-list: store the weight $w(u, v)$ of the edge $(u, v) \in E$ with vertex v in u's adjacency list.
- Adjacency-matrix: store the weight $w(u, v)$ of the edge $(u, v) \in E$ as the entry in row u and column v of the adjacency matrix. If an edge does not exist, it stores NIL, 0 or ∞ up to your convenience.

Comparision

1. Adjacency-list: requires less memory, required simply by most of the graph algorithms.
2. Adjacency-matrix: simpler and preferred when graphs are reasonably small, requires only one bit per entry.

SEARCHING A GRAPH

Searching a graph

Searching a graph means systematically following the edges of the graph so as to visit the vertices of the graph.

- A graph-searching algorithm can discover much about the structure of a graph.

Searching a graph

Searching a graph means systematically following the edges of the graph so as to visit the vertices of the graph.

- A graph-searching algorithm can discover much about the structure of a graph.
- Many algorithms begin by searching their input graph to obtain this structural information.

Searching a graph

Searching a graph means systematically following the edges of the graph so as to visit the vertices of the graph.

- A graph-searching algorithm can discover much about the structure of a graph.
- Many algorithms begin by searching their input graph to obtain this structural information.
- Techniques for searching a graph lie at the heart of the field of graph algorithms.

Breadth-first search

Definition

- Given a graph $G = (V, E)$ and a distinguished **source** vertex s, **breadth-first search** systematically explores the edges of G to *discover* every vertex that is reachable from s by computing the distance (smallest number of edges) from s to each reachable vertex.
- It also produces a **breadth-first tree** with root s that contains all reachable vertices.

Breadth-first search

Definition

- Given a graph $G = (V, E)$ and a distinguished **source** vertex s, **breadth-first search** systematically explores the edges of G to *discover* every vertex that is reachable from s by computing the distance (smallest number of edges) from s to each reachable vertex.
- It also produces a **breadth-first tree** with root s that contains all reachable vertices.

Properties:

1. For any vertex reachable from s, the simple path in the breadth-first tree from s to corresponds to a *shortest path* from s to in G, that is, a path containing the smallest number of edges.

Breadth-first search

Definition

- Given a graph $G = (V, E)$ and a distinguished **source** vertex s, **breadth-first search** systematically explores the edges of G to *discover* every vertex that is reachable from s by computing the distance (smallest number of edges) from s to each reachable vertex.
- It also produces a **breadth-first tree** with root s that contains all reachable vertices.

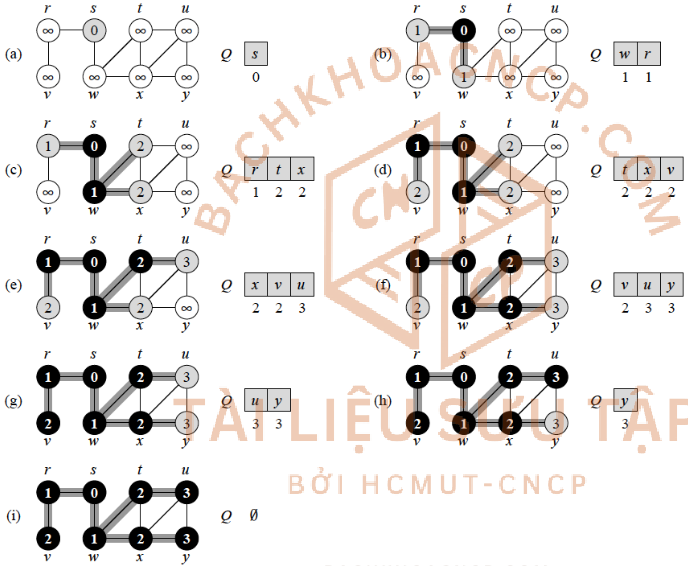
Properties:

1. For any vertex reachable from s, the simple path in the breadth-first tree from s to corresponds to a *shortest path* from s to in G, that is, a path containing the smallest number of edges.
2. Works on both directed and undirected graphs.

Breadth-first search: Pseudocode

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

Breadth-first search



(Source: Introduction to algorithms - 3rd edition)

Shortest path between two vertices

Assuming that BFS has already computed, the following procedure prints out the vertices on a shortest path from s to v:

```
PRINT-PATH(G, s, v)
if v == s
    print s
elseif v.π == NIL
    print "no path from" s "to" v "exists"
else PRINT-PATH(G, s, v.π)
    print v
```

Depth-first search

Depth-first search is the strategy to search *deeper* in the graph whenever possible.

1. Explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.

Depth-first search

Depth-first search is the strategy to search *deeper* in the graph whenever possible.

1. Explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.
2. Once all of v's edges have been explored, the search *backtracks* to explore edges leaving the vertex from which v was discovered.

Depth-first search

Depth-first search is the strategy to search *deeper* in the graph whenever possible.

1. Explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.
2. Once all of v's edges have been explored, the search *backtracks* to explore edges leaving the vertex from which v was discovered.
3. Continues until all the vertices that are reachable from the original source vertex discovered.

Depth-first search

Depth-first search is the strategy to search *deeper* in the graph whenever possible.

1. Explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.
2. Once all of v's edges have been explored, the search *backtracks* to explore edges leaving the vertex from which v was discovered.
3. Continues until all the vertices that are reachable from the original source vertex discovered.
4. If any undiscovered vertices remain, then depth-first search selects one of them as a new source, and it repeats the search from that source.

Depth-first search

Depth-first search is the strategy to search *deeper* in the graph whenever possible.

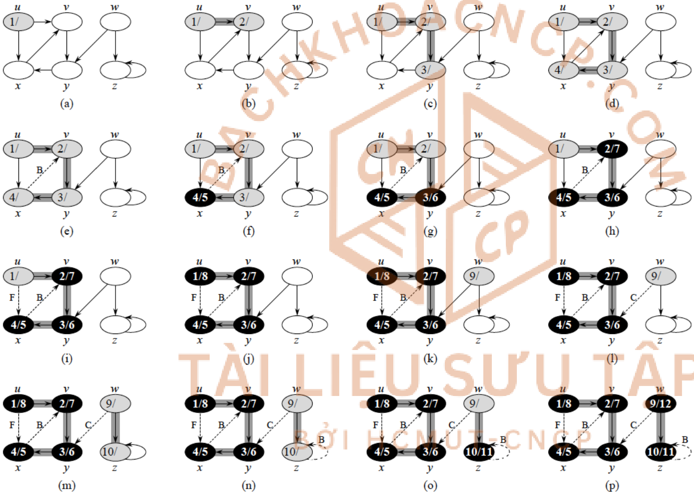
1. Explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.
2. Once all of v's edges have been explored, the search *backtracks* to explore edges leaving the vertex from which v was discovered.
3. Continues until all the vertices that are reachable from the original source vertex discovered.
4. If any undiscovered vertices remain, then depth-first search selects one of them as a new source, and it repeats the search from that source.
5. The algorithm repeats this entire process until it has discovered every vertex.

Depth-first search: Pseudocode

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

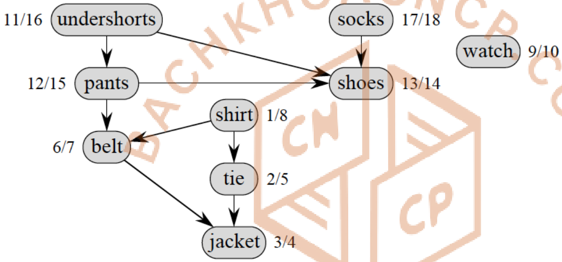
```
DFS-VISIT(G, u)
time = time + 1                 // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]           // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK                 // blacken u; it is finished
time = time + 1
u.f = time
```

Depth-first search

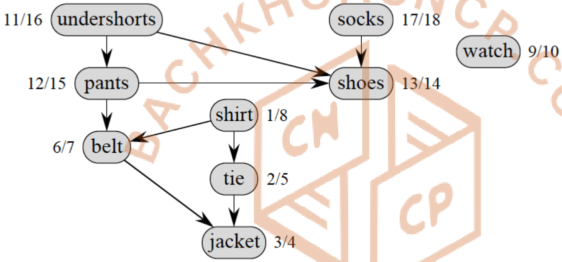


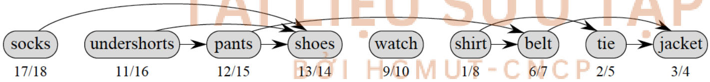
TOPOLOGICAL SORTING

Bumstead dressing



Bumstead dressing





Topological sorting

Definition

A **topological sort** of a directed acyclic graph $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v) then u appears before in the ordering. If the graph contains a cycle, then no linear ordering is possible.

Topological sorting

Definition

A **topological sort** of a directed acyclic graph $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v) then u appears before in the ordering. If the graph contains a cycle, then no linear ordering is possible.

```
TOPOLOGICAL-SORT(G)
1  call DFS(G) to compute finishing times v.f for each vertex v
2  as each vertex is finished, insert it onto the front of a linked list
3  return the linked list of vertices
```

Applications

- A common application of topological sorting is in scheduling a sequence of jobs.

Applications

- A common application of topological sorting is in scheduling a sequence of jobs.
- The jobs are represented by vertices, and there is an edge from x toy if job x must be completed before job y can be started

  For example, in constructing a building,the basement must be completed before the first floor, which must be completed before the second floor and so on.

Applications

- A common application of topological sorting is in scheduling a sequence of jobs.
- The jobs are represented by vertices, and there is an edge from x toy if job x must be completed before job y can be started

  For example, in constructing a building,the basement must be completed before the first floor, which must be completed before the second floor and so on.
- A topological sort gives an order in which we should perform the jobs.

THANK YOU.